# Deep Autoassociative Networks

Charles Hand
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California, USA
chuck@brain.jpl.nasa.gov

**Abstract:**. *Autoassociative networks are powerful and versatile information processing systems with some inconvenient limitations. Two of these limitations are the small number of patterns that can be clearly distinguished and the impoverished ability of the net to form clearly defined neighborhoods around target patterns. This paper introduces a refinement of Autoassociative networks, called 'deep' autoassociative networks. These novel networks can distinguish between large numbers of patterns, and have clearly defined basins of attraction around target patterns. The typical performance of deep autoassociative networks is significantly superior to the typical performance of standard autoassociative networks on the same tasks. Deep autoassociative networks pay for this increased level of performance by having an increased number of weights.*

***Keywords***: autoassociative nets, autonomous robots, synaptic networks

## 1. Introduction

Autoassociative networks are one of the classic artificial neural network architectures [1]. They have been used for a wide variety of pattern processing problems such as cleaning up noisy pictures and recognizing known pictures when partially occluded. They also have been applied to such important non-pattern-recognition tasks as the traveling salesman problem. Recently autoassociative networks have been applied to the control of small robots [2,3,4].

There are, however, attributes of autoassociative networks that limit their power and versatility. One of these limiting attributes is the fact that an autoassociative network can only process a limited number of patterns before these patterns start interfering with each other. Another troublesome attribute is the way autoassociative networks sometimes have trouble forming well-defined neighborhoods around target patterns. Both of these problems are dependent on the orthogonality of the target patterns; but typically, both problems severely restrict the performance of autoasssociative networks.

This paper introduces a refinement of autoassociative nets called deep autoassociative networks. These deep nets have more weights than their shallow predecessors do, but they can distinguish between many more patterns and between less orthogonal patterns. Deep nets also form more clearly defined neighborhoods (basins of attraction) than are usual for autoassociative networks.

To understand how deep autoassociative networks (DAN) differ from Standard Autoassociative Networks (SAN) we will first look at the architecture and algorithms associated with SAN and then compare these to the architecture and algorithms associated with DAN. Next we will look at how these changes affect the basins of attraction of SAN and DAN. Finally, we will show an experiment that displays the difference between the basins of attraction for SAN and DAN. In the Conclusion section, we will look at the reason DAN are particularly well suited for controlling robots.

## 1.1 Standard Autoassociative Networks

Generally speaking, autoassociative networks are a set of neurons that are fully interconnected. Each neuron has input from all other neurons, and the output of each neuron goes to every other neuron. In some instantiations, neurons also output to themselves [6]. The state of a neuron is completely determined by the dot product of its inputs and the weights associated with its input channel. Setting the weights sets the behavior of the network.

The neurons of an autoassociative network are usually thought of as comprising a row or vector. Time is a quantum phenomenon for (most) autoassociative networks in the sense that time proceeds in discrete steps or moments. At each moment of time, the row of neurons forms a pattern: Some neurons are firing, some are not firing. Hence the current state of an autoassociative network can be described with a single binary vector. As time goes by, the network changes this vector. At each timestep, a neuron is either firing (1) or not firing (0) depending on the current input from all the other neurons. Hence the matrix of weights connecting the neurons controls the movement of a vector (neuronal firing pattern) through N-space. Autoassociative networks move vectors over hyperspace landscapes of possibilities [6].

Here is an algorithm for updating the network vector (firing pattern of the neurons):

P = The current network vector
For each neuron j in the network
    Dj = the dot product of P with the input weights of neuron j.
    Use this dot product to update the firing status of neuron j
End of the loop (all neurons are updated).

The topography of trajectories through N-space is completely determined by the weight matrix. Learning consists of modifying the values of these weights. To set an autoassociative network to recall a set of patterns P1, P2, ..., Pn the weights are set to P1*P1 + P2*P2+ ... +Pn*Pn where * is the outer product of two vectors and + is matrix addition. There are also learning algorithms that set the weights incrementally.

If we look at the network from the point of view of a single weight, we start to see some of the reasons that autoassociative networks are so inefficient. Consider a network with N neurons, and look at neuron M -- or more precisely look at weight W of neuron M. Learning consists of moving W to a number that is best for most patterns. At the end of training, all weights are fixed to reflect the tyranny of the majority of patterns. All the patterns that represent minorities (from a single weight's point of view) are ignored. The performance of the network would be improved if the fixed weights were replaced with something more dynamic [7,8].

## 1.2. Deep Autoassociative Networks

In a standard autoassociative network, each neuron is again connected to all of the other neurons in the net. As in the case of SAN, at each timestep, a neuron is either firing (1) or not firing (0) depending on the current input from all the other neurons. Hence the matrix of weights connecting the neurons still controls the movement of a vector (neuronal firing pattern) through N-space. The topography of trajectories through N-space is still completely determined by the weight matrix

and learning consists of modifying the values of these weights.

However, in a deep [8] autoassociative network, each weight of a SAN is replaced with a network. SANs have a number of inputs, each of which has one weight. DANs have a number of inputs, each of which has a small network with full fan in, that only computes a dot product. Therefore, in a deep autoassociative network, the number of weights is $N^3$.

Deepening the network is analogous to adding a hidden layer of $N^2$ simple linear neurons, using a factor of N extra weights to insert them into the network. In effect, this expands the dimensionality of the embedding vector space to allow better separation of nearby states. These hidden units are much simpler than the top-level neurons in that they have only a single output rather than a full fanout.

Naturally, the updating algorithm for a DAN differs from the updating algorithm for a SAN. Here is the update algorithm for DAN:

P = The current network vector
For each neuron j in the network
    For each network k in the inputs to j
        Wk = the dot product of P with k
    End of the loop
    (W contains the vector of dot products)
    Dj = the dot product of P with W.
    Use this dot product to update the
    firing status of neuron j
End of the loop (all neurons are updated).

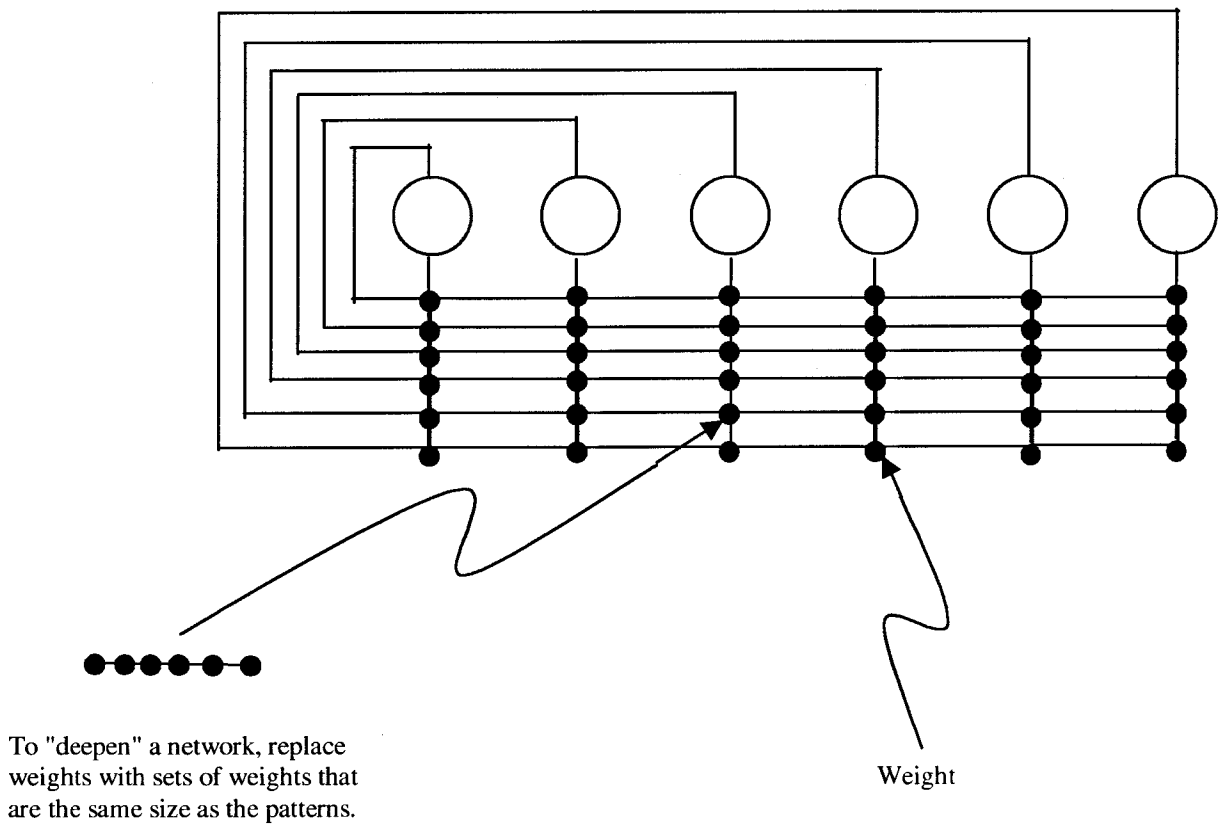


**Figure 1: An Autoassociative Network**

## 2. Basins of Attraction

The total collection of states of an autoassociative network is frequently viewed as a set of points in hyperspace. If the statesize is N, the collection of points form a landscape in N-space. Each setting of the weights produces a different landscape. The value of this visualization as an intuition aid is obvious: movement from one state to the next is always (monotonically) "downhill." This visualization reveals fixed points as the nadirs of attractor basins, and limit cycles are revealed to be the rims of flat-bottomed basins of attraction.

When an autoassociative net is used to recall a set of target patterns, the ideal topography would place each target pattern at the very bottom of a basin of attraction. Each basin in this ideal topography would include all the neighbors of a given target that were nearer to the given target than to any other target. For a net that is trained, the construction of clearly distinguishable basins is a measure of the efficiency of the training algorithm. It is in this area of basin construction that DAN is clearly superior to SAN. We are currently calculating theoretical and empirical capacity for different applications.

### 2.1 Comparative Basin Delineation

One easy way to implement a simple test of basin delineation is this:

1. Set the weights of an autoassociative network with the n patterns P1, P2, ... Pn. The weights are the sums of the outer products of each pattern with itself.

2. For each Pi, set the net to a pattern that is Hamming distance 1 from Pi, and let the net run until a fixed point (or limit cycle) is encountered.

3. Repeat step 2 for all k of the neighbors of each of the Pi (k is the pattern length). Keep track of how many of the neighbors of Pi go to Pi and how many do not.

4. If all of the neighbors behave correctly, i.e., if they all move to the attractor in one step, the neighborhood of radius 1 is a well-defined basin. The greater the number of neighbors that go astray, the poorer the basin is defined.

If necessary this test could be expanded with larger neighborhoods. The test is a standard benchmark [9] for comparing the basin formation characteristics of two different autoassociative networks. When this test is used to compare standard autoassociative networks with the deep autoassociative networks outlined in this paper, the superior basin forming ability of deep autoassociative networks is perspicuous.

In the following example (Figure 2), the states of autoassociative networks are expressed in hexadecimal numbers instead of the actual sequences of "on" and "off" states that are traditionally expressed as binary strings of ones and zeros. The enhanced readability of hexadecimal expressions is clear when the following expressions are compared:

***Hexadecimal expressions***

AAA --> BBB
and
ABC --> 123

compared to the equivalent

***Binary expressions***

101010101010 --> 101110111011

and

101010111100 --> 000100100011

In Figure 2, the trajectories of the neighbors of ABC in a standard autoassociative network are recorded. The autoassociative network, which generated the data in Figure 2, was preset with the three patterns ABC, 123, and 666.

| | | | |
|---|---|---|---|
| ABD → | ABC → | ABC → | ABC |
| ABE → | ABC → | ABC → | ABC |
| AB8 → | ABC → | ABC → | ABC |
| AB4 → | ABC → | ABC → | ABC |
| AAC → | ABC → | ABC → | ABC |
| A9C → | A9C → | A9C → | A9C |
| AFC → | ABC → | ABC → | ABC |
| A3C → | ABC → | ABC → | ABC |
| BBC → | ABC → | ABC → | ABC |
| 8BC → | ABC → | ABC → | ABC |
| EBC → | ABC → | ABC → | ABC |
| 2BC → | ABC → | ABC → | ABC |

Most of the neighbors of ABC moved to ABC on the first timestep. The neighbor A9C, however, does not go to ABC but instead acts like a fixed point. In this network there will be 4 examples where a neighbor does not go to the proper attractor. This is one of those 4 examples. When the network is required to remember the four patterns: ABC, CAB, 123, & 666, there will be 8 examples of errors.

If this standard 12X12 autoassociative network is replaced by the deep 12X12X12 network, all of the neighbors go to the proper attractors.

**Figure 2: A sample set of network runs**

## 4. Conclusions

Preliminary evidence indicates that deep autoassociative networks (DAN) have more clearly delineated basins of attraction than do standard autoassociative networks. This means that DAN can store more patterns and have less trouble recalling patterns. DAN also are less likely to confuse patterns with each other, and are more fault tolerant because small errors (neighbors) immediately repair themselves when the pattern moves to the bottom of the basin of attraction [9].

Autoassociative networks play a central role in the development of many of the newer autonomous robot brains. DAN are particularly suitable for improving the brains of small robots because of the parameters of this brain replacement. For a given robot, the length of the control vector is fixed because the control vector consists of all the robot's sensors and all the robot's effectors and this number is fixed. For small autonomous robots, we have a fixed length control vector that we would like to manipulate with greater finesse. If there is room onboard for more weights, DAN are the ideal replacement brains. As hardware gets smaller and smaller, the above argument makes more and more sense.

## 5. Acknowledgement

The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology and this work was sponsored by the National Aeronautics and Space Administration.

## 6. References

[1] D. E. Rumelhart, J. L. McLelland, "Parallel Distributed Processing", I - II, MIT Press 1986

[2] E. W. Baumann, and D. L. Williams, "Stochastic Associative Memory" in "The Science of Artificial Neural Networks II", SPIE Proceedings vol. 1966, pp. 132-139,1966

[3] D. Floreano, and F. Mondada, "Evolutionary Neurocontrollers for Autonomous Mobile Robots", Neural Networks, vol. 11, pp. 1461-1478, 1998

[4] J. Urzelai, J. Floreano, M. Dorigo, and M. Colombetti, "Incremental Robot Shaping", Connection Science, vol. 10, pp. 341-360, 1998

[5] R. M. Golden, "Mathematical Methods for Neural Network Analysis and Design," MIT Press, 1994

[6] M. Spitzer, "The Mind Within the Net", MIT Press, 1999

[7] C. Hand, "Genetic Nets," Proceedings 1997 IEEE conference on Genetic Programming, Stanford University, vol. 2, pp. 35-41, 1997

[8] C. Hand, "A Pliant Synaptic Network for Signal Analysis", The International Conference on Mathematical and Engineering Techniques in Medicine and Biological Sciences, vol. 1, pp. 275-281, METMBS Press, 2000

[9] P. Churchland, T.J. Sejnowski, "The Computational Brain", MIT Press, 1992